Research at Google

Lecture: Google **Chubby lock service**

http://research.google.com/archive/chubby.html

10/15/2014

Romain Jacotin

romain.jacotin@orange.fr

Agenda

- **Introduction**
- Design
- Mechanisms for scaling
- Experience
- Summary

The "Part-Time" Parliament
– Leslie Lamport

Introduction

Abstract

- **Chubby lock service is intended for use within a loosely-coupled distributed system consisting large number of machines (10.000) connected by a high-speed network**
  - Provides coarse-grained locking
  - And reliable (low-volume) storage
- **Chubby provides an interface much like a distributed file system with advisory locks**
  - Whole file read and writes operation (no seek)
  - Advisory locks
  - Notification of various events such as file modification
- **Design emphasis**
  - **Availability**
  - **Reliability**
  - **But not for high performance / high throughput**
- **Chubby uses asynchronous consensus: PAXOS with lease timers to ensure liveness**

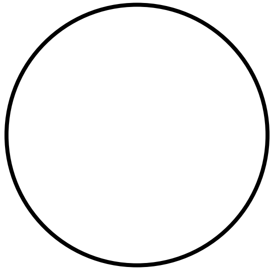
Agenda

- Introduction
- **Design**
- Mechanisms for scaling
- Experience
- Summary

“Death Star tea infuser”

Design

Google's rationale (2006)


Initial Death Star design

Design choice: Lock Service or **Client PAXOS Library ?**

- **Client PAXOS Library ?**
  - **Depend on NO other servers** (besides the name service …)
  - Provide a standard framework for programmers

- **Lock Service ?**
  - Make it easier to add availability to a prototype, and to maintain existing program structure and communication patterns
  - Reduces the number of servers on which a client depends by offering both a name service with consistent client caching and allowing clients to store and fetch small quantities of data
  - Lock-based interface is more familiar to programmers
  - Lock service use several replicas to achieve high availability (= quorums), but even a single client can obtain lock and make progress safely → **Lock service reduces the number of servers needed for a reliable client system to make progress**

Design

Google’s rationale (2006)

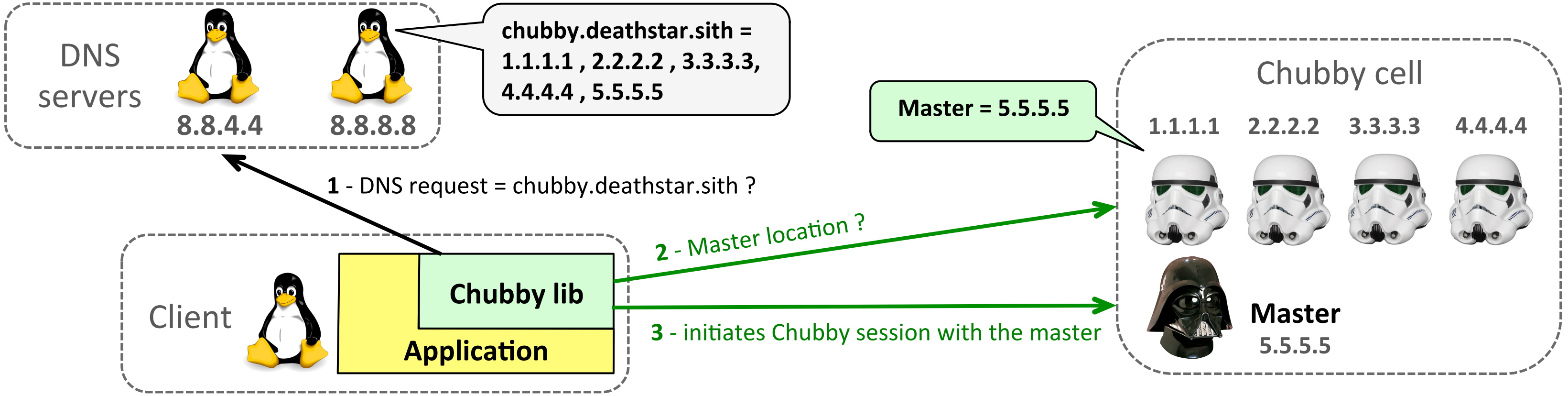
- **Two key design decisions**
  - **Google chooses lock service** (*but also provide a PAXOS client library independently from Chubby for specific projects*)
  - **Serve small files** to permit elected primaries (= client application masters) to advertise themselves and their parameters

- **Decisions based on Google's expected usage and environment**
  - **Allow thousands of clients to observe Chubby files → Events notification mechanism** to avoid polling by clients that wish to know change
  - **Consistent caching semantics** prefered by developers and caching of files protects lock service from intense polling
  - **Security mechanisms** (access control)
  - **Provide only coarse-grained locks** (long duration lock = low lock-acquisition rate = less load on the lock server)

Design

System structure

- **Two main components that communicate via RPC**
  - A replica server
  - A library linked against client applications
- **A Chubby cell consists of small set of servers (typically 5) knows as replicas**
  - Replicas use a distributed consensus protocol (**PAXOS**) to elect a master and replicate logs
  - Read and Write requests are satisfied by the master alone
  - If a master fails, other replicas run the election protocol when their master lease expire (new master elected in few seconds)
- **Clients find the master by sending master location requests to the replicas listed in the DNS**
  - Non master replicas respond by returning the identity of the master
  - Once a client has located the master, client directs all requests to it either until it ceases to respond, or until it indicates that it is no longer the master


DNS servers
8.8.4.4
8.8.8.8
chubby.deathstar.sith = 1.1.1.1 , 2.2.2.2 , 3.3.3.3, 4.4.4.4 , 5.5.5.5
Master = 5.5.5.5
Chubby cell
1.1.1.1
2.2.2.2
3.3.3.3
4.4.4.4
1 - DNS request = chubby.deathstar.sith ?
2 - Master location ?
3 - initiates Chubby session with the master
Client
Chubby lib
Application
Master
5.5.5.5

Design

PAXOS distributed consensus

- **Chubby cell with N = 3 replicas**
  - Replicas use a distributed consensus protocol to elect a master (PAXOS). **Quorum = 2 for N = 3**
  - The master must obtain votes from a majority of replicas that promise to not elect a different master for an interval of a few seconds (=master lease)
  - The master lease is periodically renewed by the replicas provided the master continues to win a majority of the vote
  - During its master lease, the master maintains copies of a simple database with replicas (ordered replicated logs)
  - Write request are propagated via the consensus protocol to all replicas (PAXOS)
  - Read requests are satisfied by the master alone
  - If a master fails, other replicas run the election protocol when their master lease expire (new master elected in few seconds)

Prepare = please votes for me and promises not to vote for someone else during 12 seconds

Promise = OK i vote for you and promise not to vote for someone else during 12 seconds

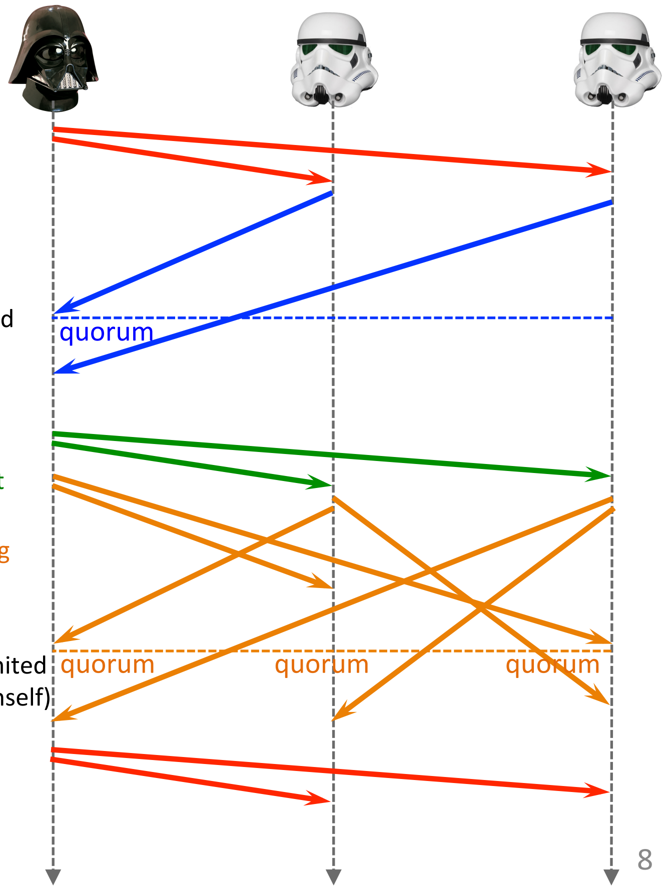
if i received quorum of **Promise** then i am the Master an i can send many **Accept** during my lease (Proposer vote for himself)

Accept = update your replicated logs with this Write client request

Accepted = i have write in my log your Write client request

if a replica received quorum of **Accepted** then the Write is commited (replica sends an Accepted to himself)

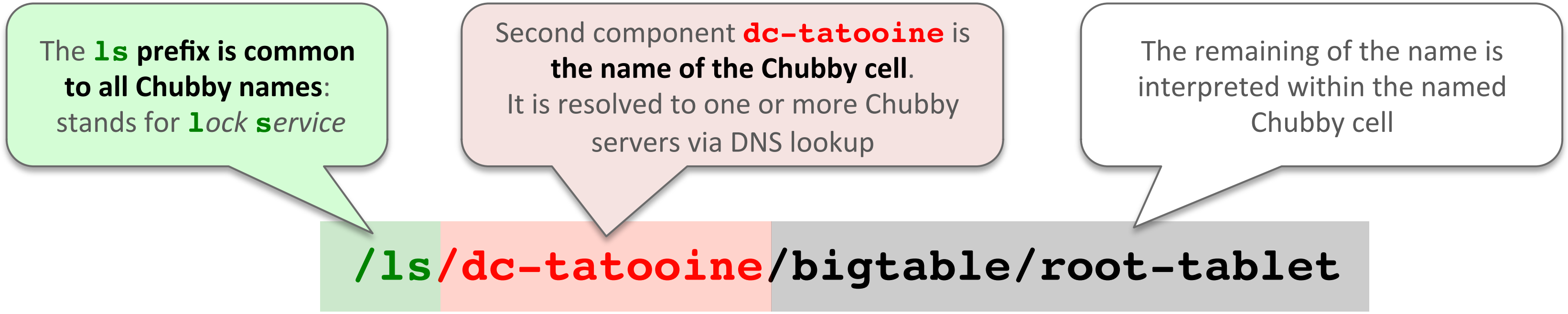
Re-Prepare = i love to be the Master, please re-votes for me before the end the lease so i can extend my lease



Design

Files & directories

- **Chubby exports a file system interface simpler than Unix**
  - Tree of files and directories with name components separated by slashes
  - Each directory contains a list of child files and directories (collectively called nodes)
  - Each file contains a sequence of un-interpreted bytes
  - **No symbolic or hard links**
  - No directory modified times, no last-access times *(to make easier to cache file meta-data)*
  - No path-dependent permission semantics: **file is controlled by the permissions on the file itself**



The `ls` **prefix is common to all Chubby names**: stands for ***l**ock **s**ervice*

Second component **`dc-tatooine`** is **the name of the Chubby cell**. It is resolved to one or more Chubby servers via DNS lookup

The remaining of the name is interpreted within the named Chubby cell

`/ls/dc-tatooine/bigtable/root-tablet`

Design

Files & directories : Nodes

- **Nodes (= files or directories) may be either permanent or ephemeral**
- **Ephemeral files are used as temporary files, and act as indicators to others that a client is alive**
- **Any nodes may be deleted explicitly**
  - Ephemeral nodes files are also deleted if no client has them open
  - Ephemeral nodes directories are also deleted if they are empty
- **Any node can act as an advisory reader/writer lock**

Design

Files & directories : **Metadata**

- **3 ACLs**
  - Three names of access control lists (ACL) used to control **reading**, **writing** and **changing the ACL names** for the node
  - Node inherits the ACL names of its parent directory on creation
  - ACLs are themselves files located in "**`/ls/dc-tatoone/acl`**" (ACL file consist of simple lists of names of principals)
  - Users are authenticated by a mechanism built into the Chubby RPC system

- **4 monotonically increasing 64-bit numbers**
  1. **Instance number**: greater than the instance number of any previous node with the same name
  2. **Content generation number** (files only): increases when the file's contents are written
  3. **Lock generation number**: increases when the node's lock transitions from ***free*** to ***held***
  4. **ACL generation number**: increases when the node's ACL names are written

- **64-bit checksum**

Design

Files & directories : Handles

- **Clients open nodes to obtain Handles** *(analogous to UNIX file descriptors)*

- **Handles include :**
  - **Check digits**: prevent clients from creating or guessing handles → *full access control checks performed only when handles are created*
  - **A sequence number**: Master can know whether a handle was generated by it or a previous master
  - **Mode information**: (provided at open time) to allow the master to recreate its state if an old handle is presented to a newly restarted master

Design

Locks, Sequencers and Lock-delay

- **Each Chubby file and directory can act as a reader-writer lock** (locks are advisory)

- **Acquiring a lock in either mode requires write permission**
  - **Exclusive mode** (writer): One client may hold the lock
  - **Shared mode** (reader): Any number of client handles may hold the lock

- **Lock holder can request a Sequencer :** opaque byte string describing the state of the lock immediately after acquisition
  - Name of the lock + Lock mode (exclusive or shared) + Lock generation number

- **Sequencer usage**
  - Application's master can generate a sequencer and send it with any internal order sends to other servers
  - Application's servers that receive orders from a master can check with Chubby if the sequencer is still good (= not a stale master)

- **Lock-delay : Lock server prevents other clients from claiming the lock during lock-delay period if lock becomes free**
  - client may specify any look-delay up to 60 seconds
  - This limit prevents a faulty client from making a lock unavailable for an arbitrary long time
  - Lock delay protects unmodified servers and clients from everyday problems caused by message delays and restarts …

Design

Events

- **Session events can be received by application**
  - **Jeopardy**: when session lease timeout and Grace period begins (see Fail-over later ;-)
  - **Safe**: when the session is known to have survived a communication problem
  - **Expired**: if the session timeout

- **Handle events: clients may subscribe to a range of events when they create a Handle (=Open phase)**
  - **File contents modified**
  - **Child node added/removed/modified**
  - **Master failed over**
  - **A Handle (and it's lock) has become invalid**
  - **Lock acquired**
  - **Conflicting lock request from another client**

- **These events are delivered to the clients asynchronously via an up-call from the Chubby library**

- **Mike Burrows:** *"The last two events mentioned are rarely used, and with hindsight could have been omitted."*

Design

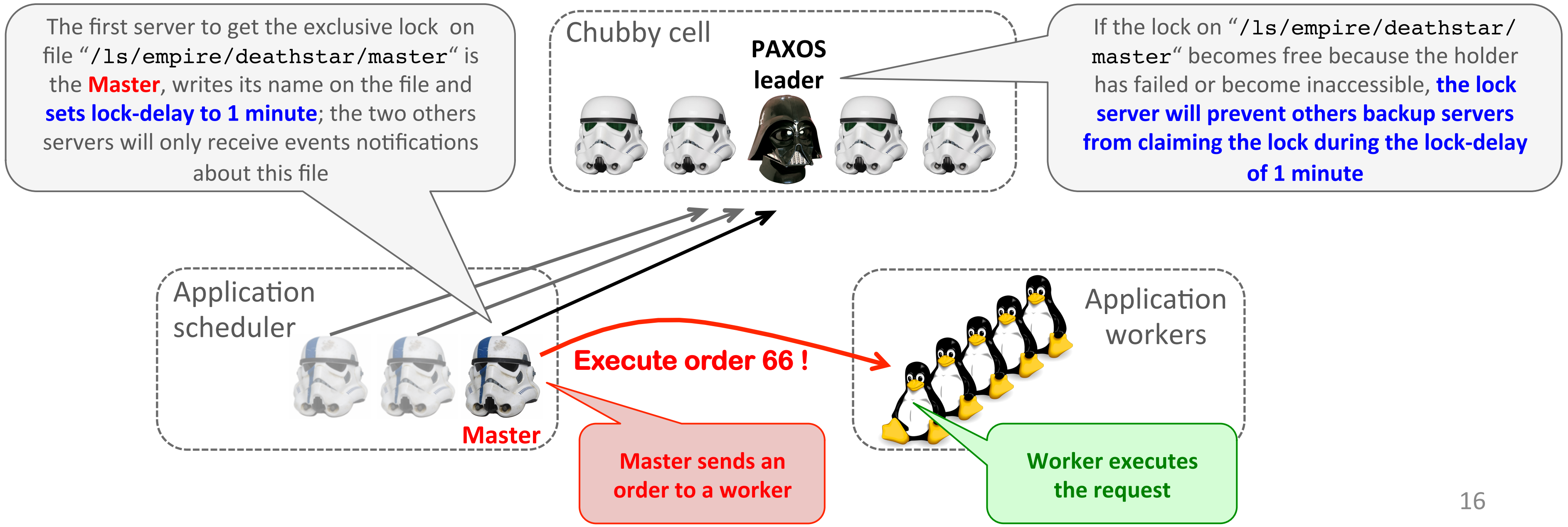
API

- Open/Close node name
  - func **Open( )** Handles are created by Open() and destroyed by Close()
  - func **Close( )** This call never failed
- Poison
  - func **Poison( )** Allows a client to cancel Chubby calls made by other threads without fear of de-allocating the memory being accessed by them
- Read/Write full contents
  - func **GetContentsAndStat( )** Atomic reading of the entire content and metadata
  - func **GetStat( )** Reading of the metadata only
  - func **ReadDir( )** Reading of names and metadata of the directory
  - func **SetContents()** Atomic writing of the entire content
- ACL
  - func **SetACL( )** Change ACL for a node
- Delete node
  - func **Delete( )** If it has no children
- Lock
  - func **Acquire( )** Acquire a lock
  - func **TryAcquire( )** Try to acquire a potentially conflicting lock by sending “conflicting lock request” to the holder
  - func **Release( )** Release a lock
- Sequencer
  - func **SetSequencer( )** Returns a sequencer that describes any lock held by this Handle
  - func **GetSequencer( )** Associate a sequencer with a Handle. Subsequent operations on the Handle failed if the sequencer is no longer valid
  - func **CheckSequencer( )** Checks whether a sequencer is valid

Design

Design n°1

- Primary election example without sequencer usage, but with lock-delay (**worst design**)



The first server to get the exclusive lock on file "`/ls/empire/deathstar/master`" is the **Master**, writes its name on the file and **sets lock-delay to 1 minute**; the two others servers will only receive events notifications about this file

If the lock on "`/ls/empire/deathstar/master`" becomes free because the holder has failed or become inaccessible, **the lock server will prevent others backup servers from claiming the lock during the lock-delay of 1 minute**

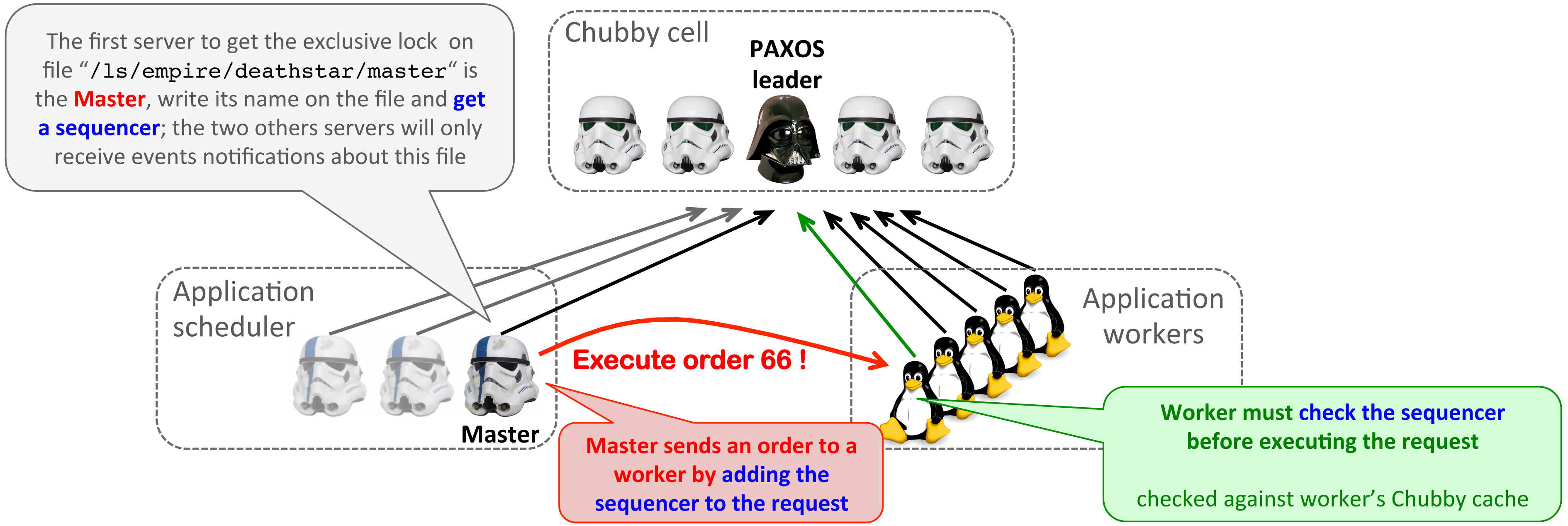
Master sends an order to a worker

Worker executes the request

Design

Design n°2

- Primary election example with sequencer usage (**best design**)



The first server to get the exclusive lock on file "`/ls/empire/deathstar/master`" is the **Master**, write its name on the file and **get a sequencer**; the two others servers will only receive events notifications about this file

Master sends an order to a worker by adding the sequencer to the request

Worker must check the sequencer before executing the request

checked against worker's Chubby cache

Design

Design n°3

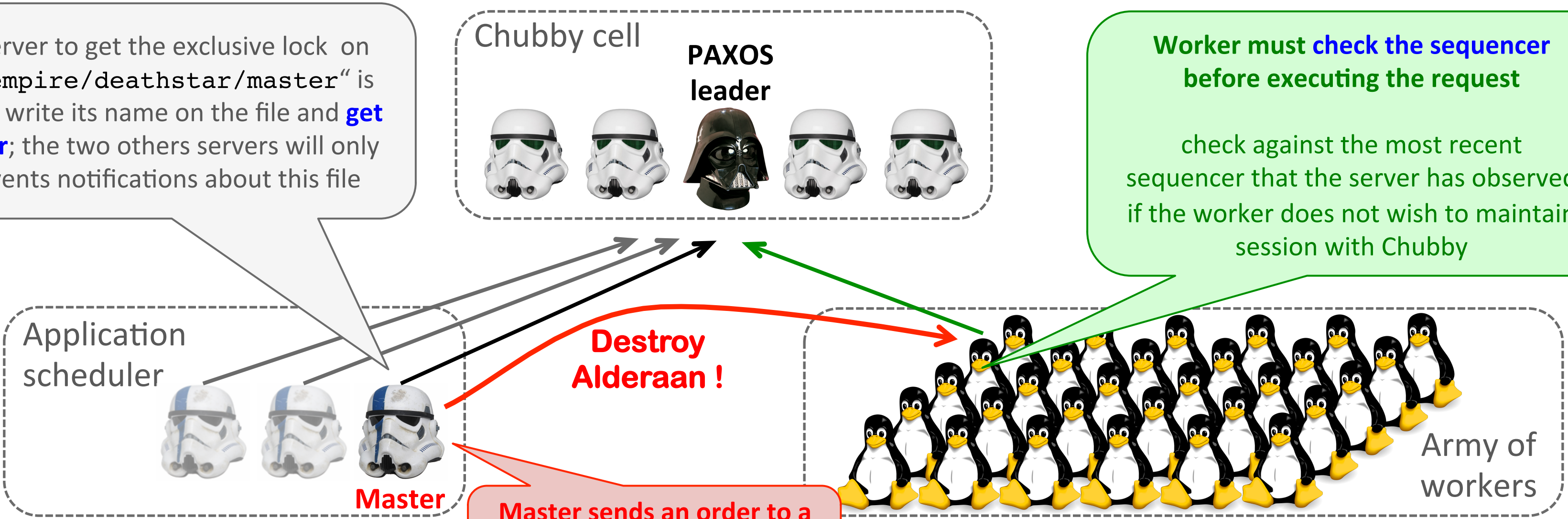
- Primary election example with sequencer usage (**optimized design**)

The first server to get the exclusive lock on file "`/ls/empire/deathstar/master`" is the **Master**, write its name on the file and **get a sequencer**; the two others servers will only receive events notifications about this file

Worker must check the sequencer before executing the request

check against the most recent sequencer that the server has observed if the worker does not wish to maintain session with Chubby

Master sends an order to a worker by adding the sequencer to the request



Design

Caching

- **Clients cache file data and node metadata in a consistent, write-through in memory cache**
  - **Cache maintained by a lease mechanism** (client that allowed its cache lease to expire then informs the lock server)
  - **Cache kept consistent by invalidations sent by the master**, which keeps a list of what each client may be caching
- **When file data or metadata is to be changed**
  - Master block modification while sending invalidations for the data to every client that may cached it
  - Client that receives of an invalidation flushes the invalidated state and acknowledges by making its next KeepAlive call
  - The modification proceeds only after the server knows that each client has invalidated its cache, either by acknowledged the invalidation, or because the client allowed its cache lease to expire

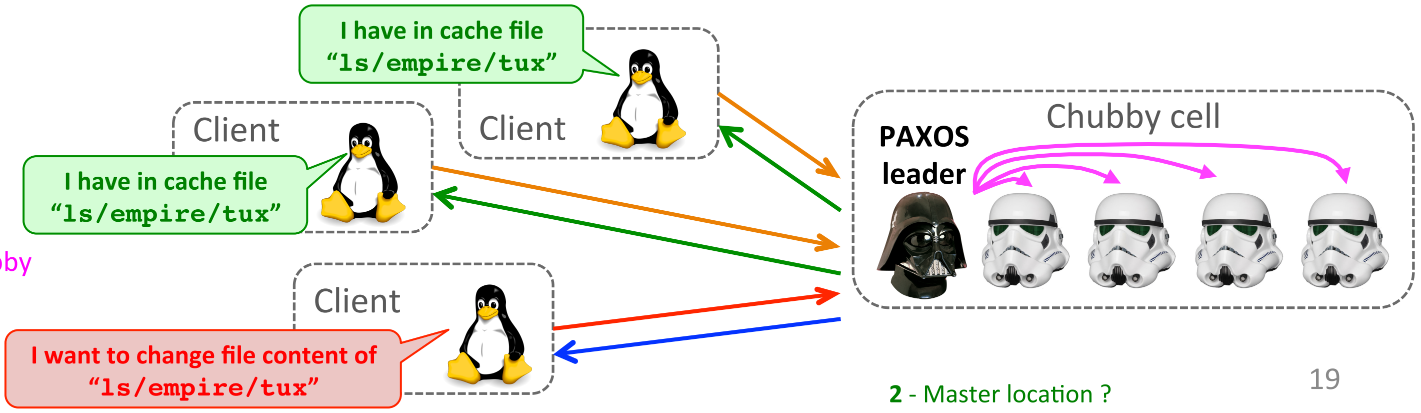
→ Request for changing content of file “X”

→ Master sends cache invalidation for clients that cache “X”

→ Clients flushes caches for file “X” and acknowledge the Master

→ Write change to the Chubby replicated database

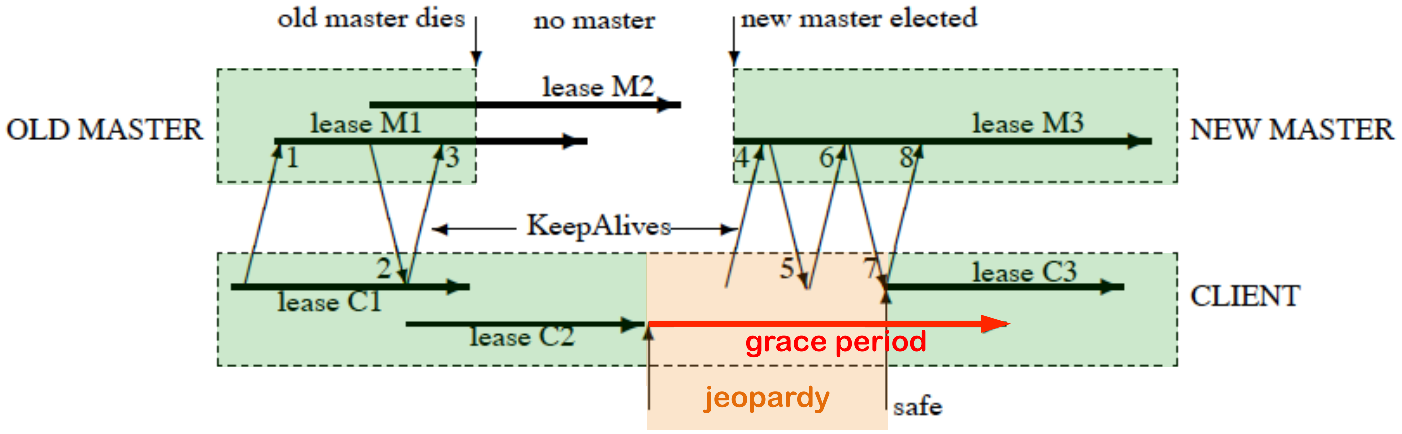
→ Master acknowledges the writer about change done


I have in cache file “ls/empire/tux”
Client
Client
I have in cache file “ls/empire/tux”
Chubby cell
PAXOS leader
Client
I want to change file content of “ls/empire/tux”

Design

Sessions and KeepAlives : Handles, locks, and cached data all remain valid while session remains valid

- **Client requests a new session on first contacting the master of a Chubby cell**
- **Client ends the session explicitly either when it terminates, or if the session has been idle**
  - **Session idle** = no opens handles and no call for a minute
- **Each session has an associated lease interval**
  - **Lease interval** = master guarantees not to terminate the session unilaterally until the lease timeout
  - **Lease timeout** = end of the lease interval

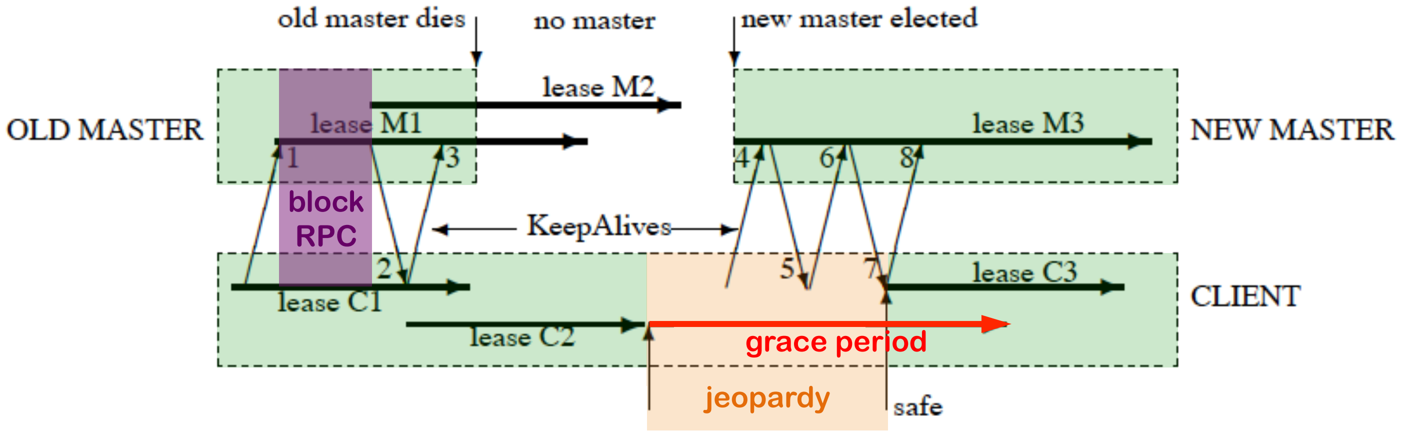


Design

Sessions and KeepAlives

- Master advances the lease timeout
  - On session creation
  - When a master fail-over occurs
  - When it responds to a KeepAlive RPC from the client:
    - On receiving a KeepAlive (1), Master blocks the RPC until client's previous lease interval is close to expiring
    - Master later allows the RPC to return to the client (2) and informs it about the new lease timeout (= lease M2)
    - Master use default extension of 12 second, overload master may use higher values
    - Client initiates a KeepAlive immediately after receiving the previous reply

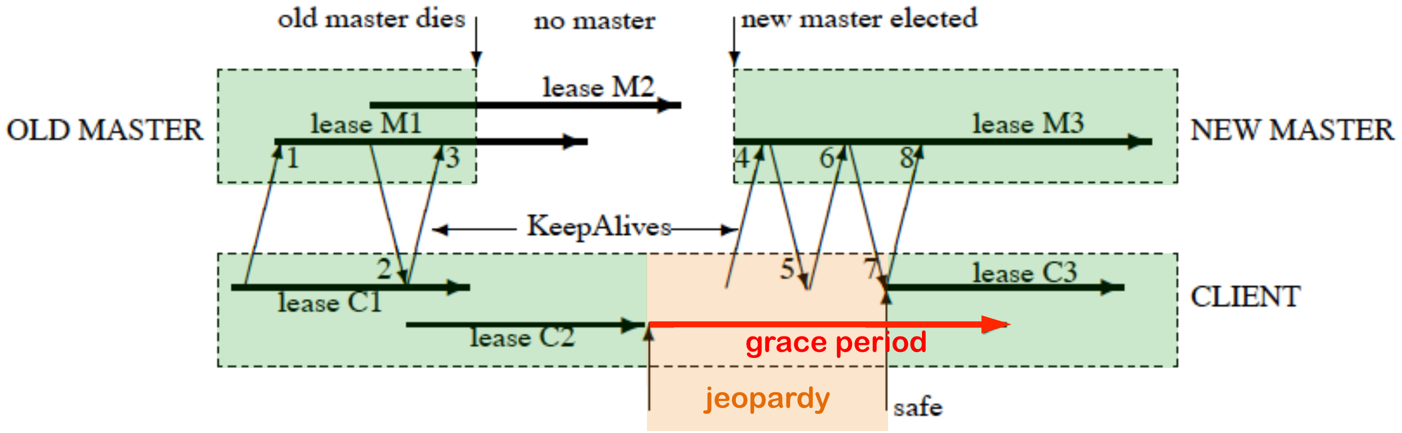
Protocol optimization : KeepAlive reply is used to transmit events and cache invalidations back to the client



Design

Fail-overs

- When a master fails or otherwise loses mastership
  - Discards its in-memory state about sessions, handles, and locks
  - Session lease timer is stopped
  - If a master election occurs quickly, clients can contact the new master before their local lease timer expire
  - If the election takes a long time, client flush their caches (= JEOPARDY) and wait for the GRACE PERIOD (45 seconds) while trying to find the new master



Design

Fail-overs : **Newly elected master's tasks** `(initial design)`

1. **Picks a new client epoch number** (clients are required to present on every call)
2. **Respond to master-location requests**, but does not at first process incoming session-related operations
3. **Builds in-memory data structures for sessions and locks recorded in the database**. Session leases are extended to the maximum that the previous master may have been using
4. **Lets clients perform KeepAlives**, but no other session-related operations
5. **Emits a fail-over event to each session**: clients flush their caches and warn applications that other events may have been lost
6. **Waits until each session expire or acknowledges the fail-over event**
7. **Now, allows all operations to proceed**
8. **If a client uses a handle created prior to the fail-over, the Master recreates the in-memory representation of the handle and then honors the call**
9. **After some interval** (1 minute)**, master deletes ephemeral files that have no open file handles**: clients should refresh handles on ephemeral files during this interval after a fail-over

Design

Database implementation

- Simple key/value database using write ahead logging and snapshotting
- Chubby needs atomic operations only (no general transactions)
- Database log is distributed among the replicas using a distributed consensus protocol (PAXOS)

Backup

- Every few hours, Master writes a snapshot of its database to **a GFS file server in a different building = no cyclic dependencies** (because local GFS uses local Chubby cell …)
- Backup used for disaster recovery
- Backup used for initializing the database of a newly replaced replica = no load on others replicas

Design

Mirroring

- **Chubby allows a collection of files to be mirrored from one cell to another**
  - Mirroring is fast: small files and the event mechanism to inform immediately if a file is added, deleted, or modified
  - Unreachable mirror remains unchanged until connectivity is restored: updated files identified by comparing checksums
  - Used to copy configuration files to various computing clusters distributed around the world

- **A special "global" Chubby cell**
  - Subtree "**`/ls/global/master`**" mirrored to the subtree "**`/ls/cell/slave`**" in every other Chubby cell
  - The "global" cell replicas are located in widely-separated parts of the world
  - Usage:
    - Chubby's own ACLs
    - Various files in which Chubby cells and other systems advertise their presence to monitoring services
    - Pointers to allow clients to locate large data sets (Bigtable cells) and many configurations files

Agenda

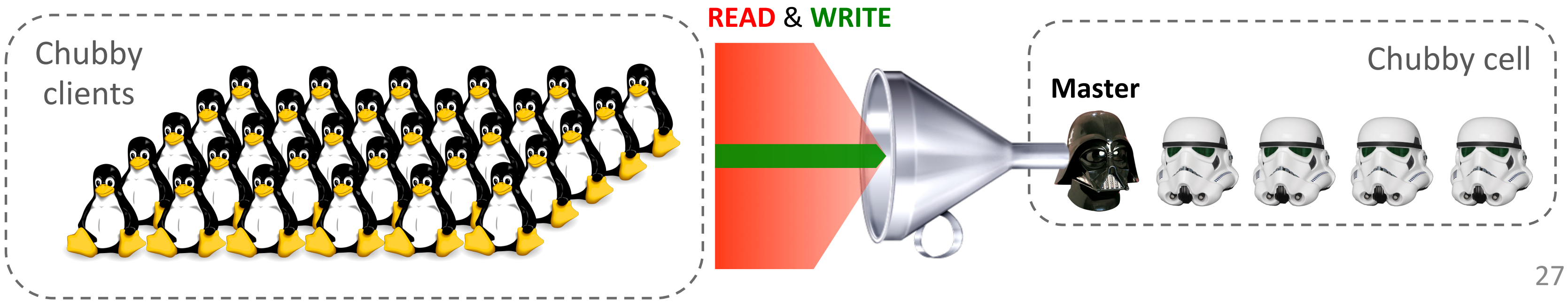
- Introduction
- Design
- **Mechanisms for scaling**
- Experience
- Summary

"Judge me by my size, do you ?" - Yoda

Mechanisms for scaling

+ 90.000 clients communicating with a Chubby master !

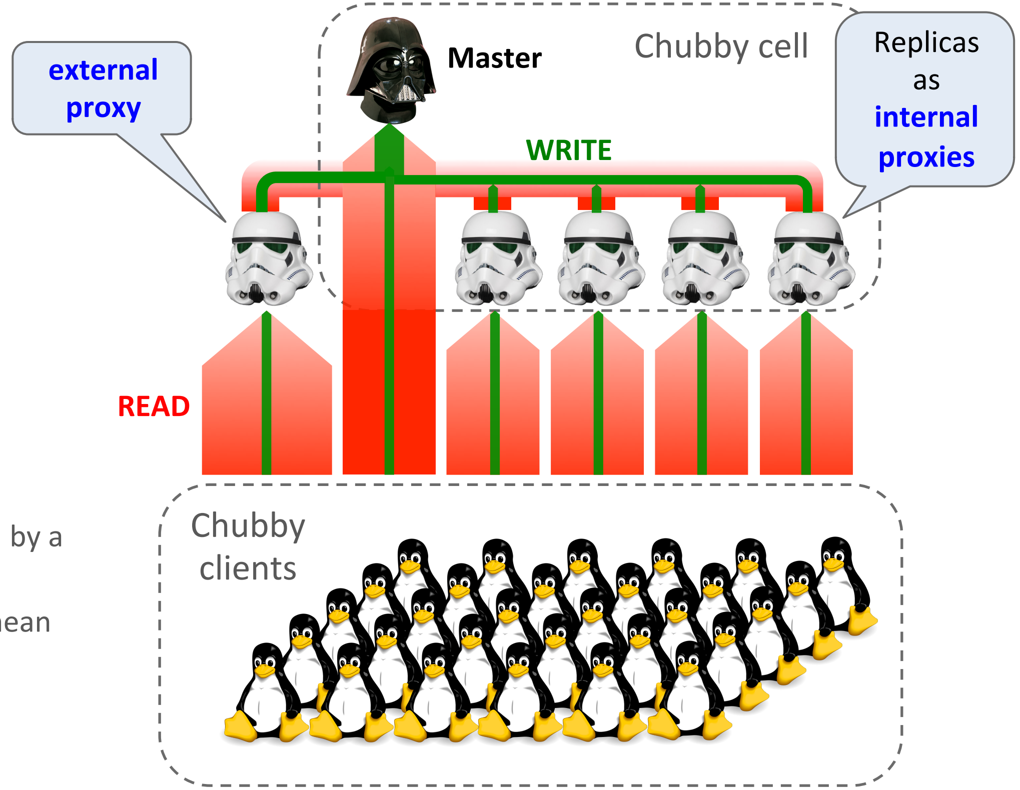
- **Chubby's clients are individual processes → Chubby handles more clients than expected**
- **Effective scaling techniques = reduce communication with the master**
  - **Minimize RTT**: Arbitrary number of Chubby cells: clients almost always use nearby cell (found with DNS) to avoid reliance on remotes machine (= 1 chubby cell in a datacenter for several thousand machines)
  - **Minimize KeepAlives load**: Increase lease times from the default=12s up to around 60s under heavy load (= fewer KeepAlive RPC to process)
  - **Optimized caching**: Chubby clients cache file data, metadata, absence of files, and open handles
  - **Protocol-conversion servers**: Servers that translate the Chubby protocol into less-complex protocols (DNS, …)



Mechanisms for scaling

Proxies

- Chubby's protocol can be proxied
  - Same protocol on both sides
  - Proxy can reduce server load by handling both KeepAlive and read requests
  - Proxy cannot reduce write traffic
- Proxies allow a significant increase in the number of clients
  - If proxy handle N clients, KeepAlive traffic is reduced by a factor of N (could be 10.000 or more ! ☺)
  - Proxy cache can reduce read traffic by at most the mean amount of read-sharing



Mechanisms for scaling

Partitioning (Intended to enable large Chubby cells with little communication between the partitions)

- **The code can partition the namespace by directory**
  - Chubby cell = **`N`** **partitions**
  - Each partition has a set of replicas and a master
  - Every **node `D/C`** in **directory `D`** would be stored on the **partition `P(D/C) = hash(D)` mod N**
  - Metadata for **`D`** may be stored on a different partition **`P(D) = hash(D') mod N`**, where **`D'` is the parent of `D`**
  - Few operations still require cross-partition communication
    - ACL: one partition may use another for permissions checks (only `Open()` and `Delete()` calls requires ACL checks)
    - When a directory is deleted: a cross-partition call may be needed to ensure that the directory is empty
- **Unless number of partitions N is large, each client would contact the majority of the partitions**
  - **Partitioning reduces read and write traffic on any given partition by a factor of N** ☺
  - **Partitioning does not necessarily reduce KeepAlive traffic …** ☹

- *Partitioning implemented in the code but not activated because Google does not need it (2006)*

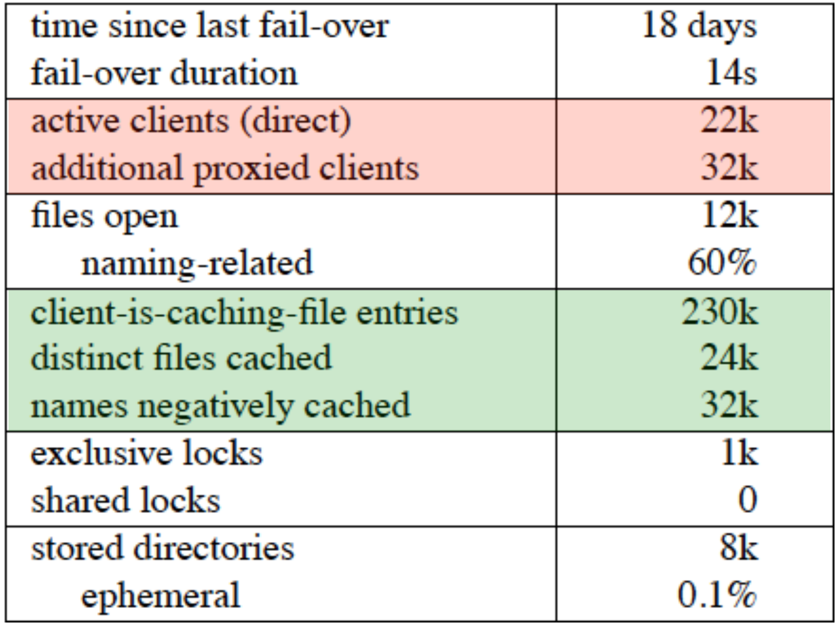
Agenda

- Introduction
- Design
- Mechanisms for scal
- **Experience**
- Summary

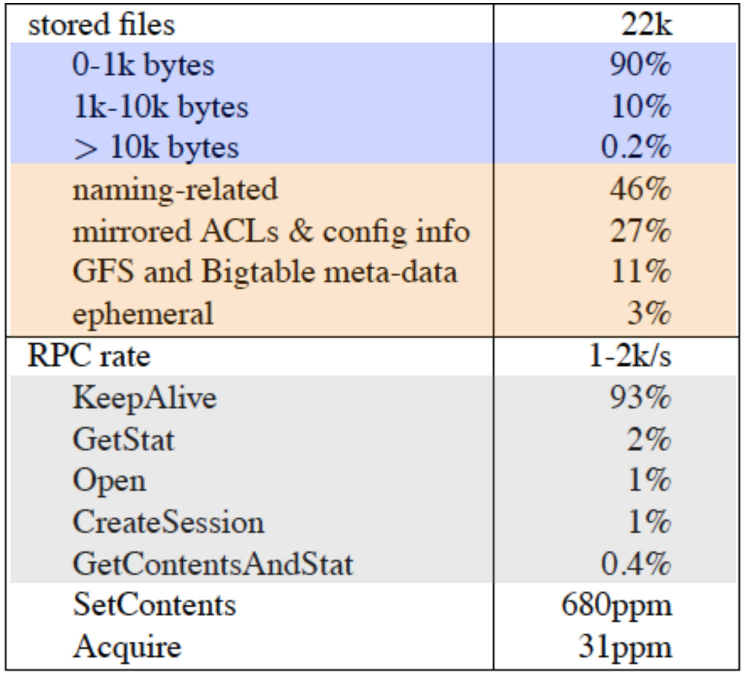
“Do. Or do not. There is no try.” - Yoda

Experience (2006)

Use and behavior : Statistics taken as a snapshot of a Chubby cell (RPC rate over 10 minutes period)



time since last fail-over	18 days
fail-over duration	14s
active clients (direct)	22k
additional proxied clients	32k
files open	12k
naming-related	60%
client-is-caching-file entries	230k
distinct files cached	24k
names negatively cached	32k
exclusive locks	1k
shared locks	0
stored directories	8k
ephemeral	0.1%



stored files	22k
0-1k bytes	90%
1k-10k bytes	10%
> 10k bytes	0.2%
naming-related	46%
mirrored ACLs & config info	27%
GFS and Bigtable meta-data	11%
ephemeral	3%
RPC rate	1-2k/s
KeepAlive	93%
GetStat	2%
Open	1%
CreateSession	1%
GetContentsAndStat	0.4%
SetContents	680ppm
Acquire	31ppm

Experience (2006)

Use and behavior

Typical causes of outages

- **61 outages over a period of a few weeks amounting to 700 cell-days of data in total**
  - **52 outages < 30 seconds → most applications are not affected significantly by Chubby outages under 30 seconds**
  - **4** caused by network maintenance
  - **2** caused by suspected network connectivity problems
  - **2** caused by software errors
  - **1** caused by overload

Few dozens cell-years of operation

- Data lost on 6 occasions
  - **4** database errors
  - **2** operator error

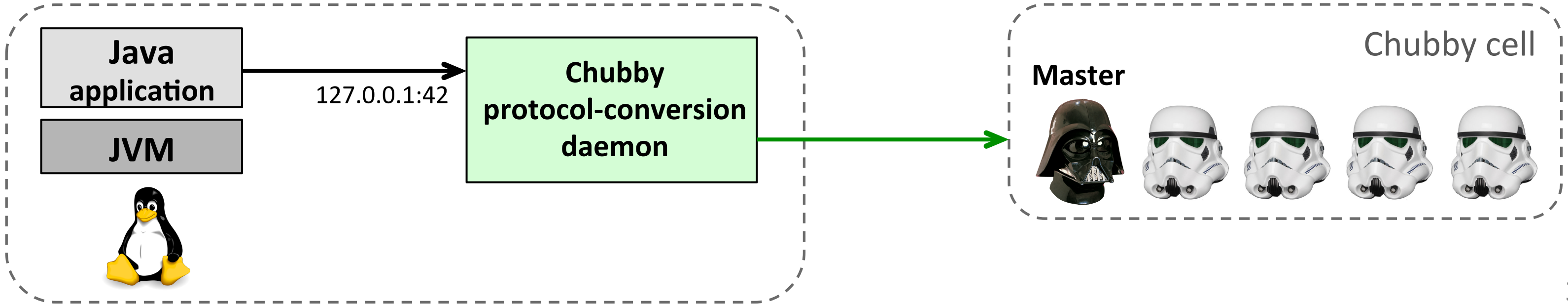
Overload

- Typically occurs when more than 90.000 sessions are active or simultaneous millions of reads

Experience (2006)

Java clients

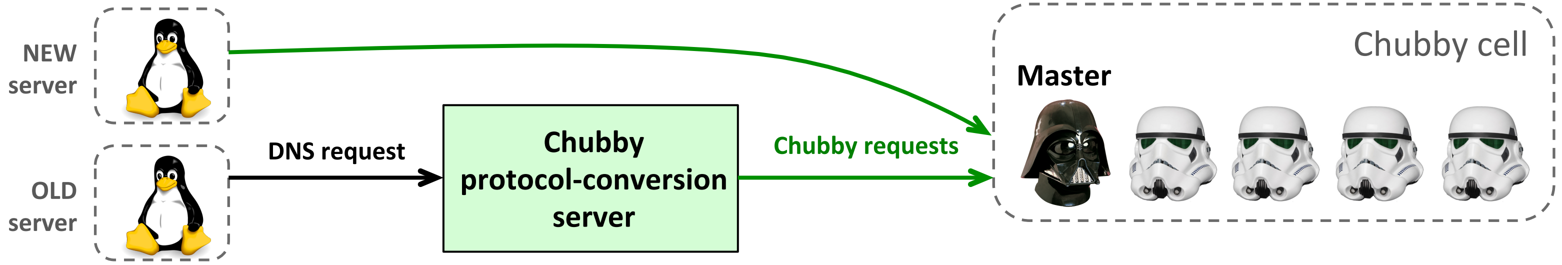
- **Chubby is in C++ like most Google's infrastructure**
- **Problem**: a growing number of systems are being written in Java
  - Java programmers dislike Java Native Interface (slow and cumbersome) for accessing non-native libraries
  - Chubby's C++ client library is 7.000 lines : maintaining a Java library version is delicate and too expensive
- **Solution**: Java users run copies of a protocol-conversion server that exports a simple RPC protocol that correspond closely to Chubby's client API
- **Mike Burrown** (2006): *"Even with hindsight, it is not obvious how we might have avoided the cost of writing, running and maintaining this additional server"*



Experience (2006)

Use as a name service

- Chubby was designed as a lock service, but popular use was as a name server
- DNS caching is based on time
  - Inconsistent caching even with small TTL = DNS data discarded when not refreshed within TTL period
  - A low TTL overloads DNS servers
- Chubby caching use explicit invalidations
  - Consistent caching
  - No polling
- Chubby DNS server
  - Another protocol-conversion server that makes the naming data stored within Chubby available to DNS clients: for easing the transition from DNS names to Chubby names, and to accommodate existing applications that cannot be converted easily such as browsers

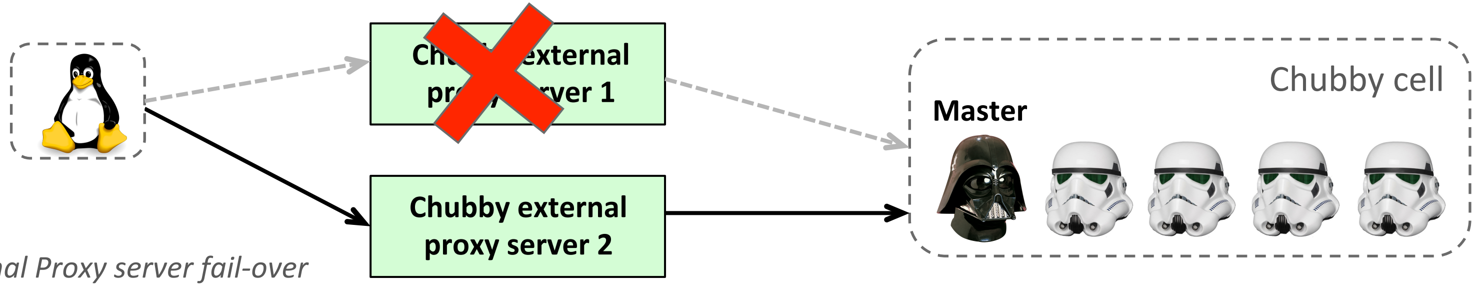


Experience (2006)

Problems with fail-over

- **Original design requires master to write new sessions to the database as they are created**
  - **Overhead on Berkeley DB version of the lock server !!!**

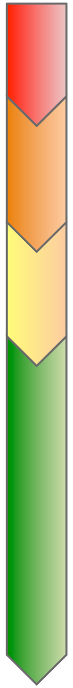
- **New design avoid recording sessions in the database**
  - **Recreate sessions in the same way the master currently recreates Handles → new elected master'task n°8**
  - **A new master must now wait a full worst-case lease-timeout before allowing operations to proceed**
    - It cannot know whether all sessions have checked in → **new elected master'task n°6**
  - **Proxy fail-over made possible because proxy servers can now manage sessions that the master is not aware of**
    - Extra operation available only on trusted proxy servers to take over a client from another when a proxy fails


Chubby external proxy server 1
Chubby external proxy server 2
Chubby cell
Master

Example: external Proxy server fail-over

Design

Problems with fail-over : Newly elected master's tasks (New design)



1. **Picks a new client epoch number** (clients are required to present on every call)
2. **Respond to master-location requests**, but does not at first process incoming session-related operations
3. **Builds in-memory data structures for ~~sessions and~~ locks recorded in the database**. ~~Session leases are extended to the maximum that the previous master may have been using~~
4. **Lets clients perform KeepAlives, but no other session-related operations**
5. **Emits a fail-over event to each session**: clients flush their caches and warn applications that other events may have been lost
6. **Waits until each session expire** or ~~acknowledges the fail-over event~~
7. **Now, allows all operations to proceed**
8. **If a client uses a handle created prior to the fail-over, the Master recreates the in-memory representation of the session and the handle and then honors the call**
9. **After some interval** (1 minute)**, master deletes ephemeral files that have no open file handles**: clients should refresh handles on ephemeral files during this interval after a fail-over

Experience (2006)

Abusive clients

- Many services use shared Chubby cells: need to isolate clients from the misbehavior of others

Problems encountered:

1. **Lack of aggressive caching**
   - **Developers regularly write loops that retry indefinitely when a file is not present, or poll a file by opening it and closing it repeatedly**
   - **Need to cache the absence of file and to reuse open file handles**
   - Requires to spend more time on DEVOPS education but in the end it was easier to make repeated `Open()` calls cheap …

2. **Lack of quotas**
   - **Chubby was never intended to be used as a storage system for large amounts of data !**
   - **File size limit introduced: 256 KB**

3. **Publish/subscribe**
   - **Chubby design not made for using its event mechanisms as a publish/subscribe system !**
   - **Project review about Chubby usage and growth predictions (RPC rate, disk space, number of files)** → need to track the bad usages …

Experience (2006)

Lessons learned

- **Developers rarely consider availability**
  - Inclined to treat a service like Chubby like as though it were always available
  - Fail to appreciate the difference between a service group up and that service being available to their applications
  - API choices can affect the way developers chose to handle Chubby outages: many DEVs chose to crash theirs apps when a master fail-over take place, but the first intent was for clients to check for possible changes …

- **3 mechanisms to prevent DEVs from being over-optimistic about Chubby availability**
  1. Project review
  2. Supply libraries that perform high-level tasks so that DEVs are automatically isolated from Chubby outages
  3. Post-mortem of each Chubby outage: eliminates bugs in Chubby and Ops precedure+ reducing Apps sensitivity to Chubby's availability

Experience (2006)

Opportunities design changes

- **Fine grained locking could be ignored**
  - DEVs must remove unnecessary communication to optimize their Apps → finding a way to use coarse-grained locking

- **Poor API choice have unexpected affects**
  - **One mistake**: means for cancelling long-running calls are the `Close()` and `Poison()` RPCs, which also discard the server state for the handle … → **may add a `Cancel()` RPC to allow more sharing of open handles**

- **RPC use affects transport protocols**
  - KeepAlives used both for refreshing the client's lease, and for passing events and cache invalidations from the master to the client: TCP's back off policies pay no attention to higher-level timeouts such as Chubby leases, so **TCP-based KeepAlives led to many lost sessions at time of high network congestion → forced to send KeepAlive RPCs via UDP rather than TCP …**
  - **May augment the protocol with an additional TCP-based `GetEvent()` RPC which would be used to communicate events and invalidations in the normal case, used in the same way KeepAlives. KeepAlive reply would still contain a list of unacknowledged events so that events must eventually be acknowledged**

Agenda

- Introduction
- Design
- Mechanisms for scaling
- Experience
- **Summary**

"May the lock service be with you."

Summary

Chubby lock service

- **Chubby is a distributed lock service for coarse-grained synchronization of distributed systems**
  - Distributed consensus among few replicas for fault-tolerance
  - Consistent client-side caching
  - Timely notification of updates
  - Familiar file system interface
- **Become the primary Google internal name service**
  - Common rendez-vous mechanism for systems such as MapReduce
  - To elect a primary from redundant replicas (GFS and Bigtable)
  - Standard repository for files that require high-availability (ACLs)
  - Well-known and available location to store a small amount of meta-data (= root of the distributed data structures)
- **Bigtable usage**
  - To elect a master
  - To allow the master to discover the servers its controls
  - To permit clients to find the master

Any questions ?

Romain Jacotin

romain.jacotin@orange.fr